Elementy projektowania inżynierskiego

Rozwiązanie problemu pręta rozciąganego metodą MES

Uwagi:

- Instrukcja została opracowana z wykorzystaniem języka *Python* w wersji *3.12.2*, dodatkowo wymagane są biblioteki *NumPy* (użyto wersji *1.26.3*) i *Matplotlib* (użyto wersji *3.8.2*). W pracowniach komputerowych lub na klastrze komputerowym mogą być zainstalowane inne wersje oprogramowania.
- Wygodę pracy znacznie podnosi praca w dowolnym środowisku deweloperskim przeznaczonym do języka Python lub w środowisku przeznaczonym specjalnie do obliczeń naukowych (np. *Spyder*).
- Wszystkie powyżej wymienione elementy można znaleźć w dystrybucjach Pythona przeznaczonych dla obliczeń naukowych, np. *WinPython* (https://winpython.github.io).
- W środowisku Spyder generowane wykresy pokazują się na panelu **Plots**, można wymusić rysowanie wykresów w oknie za pomocą poleceń:

```
from IPython import get_ipython
get_ipython().run_line_magic("matplotlib", "qt")
```

- Ponowne włączenie panelu **Plots**:

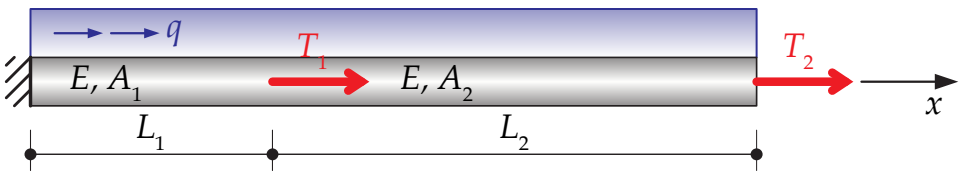
```
from IPython import get_ipython
get_ipython().run_line_magic("matplotlib", "inline")
```

- Zmiana sposobu wyświetlania tablic:

```
np.set_printoptions(formatter={"float": lambda x: f"{x:10.4g}"}, suppress = True)
```

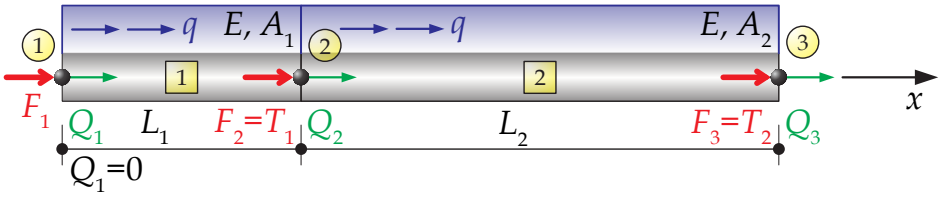
- W zaprezentowanym skrypcie niektóre obliczane zmienne są wyświetlane w konsoli **IPtyton** w trakcie wykonywania (polecenia **`print()`**). Zmienne można także oglądać w panelu **Variable Explorer**.
- W Pythonie elementy tablic i list numeruje się od 0 (zera).

Pręt rozciągany



- $L_1 = 0{,}4$ m
- $L_2 = 1{,}2$ m
- $A_1 = 0{,}5 \cdot 10^{-3}$ m
- $A_2 = 0{,}4 \cdot 10^{-3}$ m
- $E = 200$ GPa
- $q = 10$ kN/m
- $T_1 = 40$ kN
- $T_2 = 5$ kN

Dyskretyzacja pręta



Import bibliotek i dodatkowe ustawienia

```
import numpy as np
import matplotlib as mpl
from IPython import get_ipython

get_ipython().run_line_magic("matplotlib", "qt")
np.set_printoptions(formatter={"float": lambda x: f"{x:10.4g}"}, suppress = True)
```

Definicja macierzy sztywności i wektora równoważników węzłowych obciążenia

- Są to zwykłe funkcje języka Python, które zwracają tablice typu **`array`** z biblioteki NumPy.

```
def Ke(EA, L):
  return np.array([[ EA / L, -EA / L ],
                   [-EA / L,  EA / L ]])

def Pe(q, L):
  return np.array([[ q * L/2 ],
                   [ q * L/2 ]])
```

Dane geometryczne, materiałowe i obciążenie

```
L1 = 0.4
A1 = 0.5e-3

L2 = 0.8
A2 = 0.4e-3

q = 10e3

T1 = 40e3
T2 = 5e3

E = 200e9
```

Liczba węzłów modelu, liczba węzłów elementu, liczba stopni swobody w węźle elementu

```
ln = 3
lw = 2
lq = 1
```

Budowa i wyświetlenie macierzy Boole'a

```
B1 = np.zeros((lw * lq, ln * lq))
B2 = np.zeros((lw * lq, ln * lq))

print(f"{B1} = B1", end="\n\n")
print(f"{B2} = B2", end="\n\n")

B1[[0, 1], [0, 1]] = 1
B2[[0, 1], [1, 2]] = 1
```

```
print(f"{B1} = B1", end="\n\n")
print(f"{B2} = B2", end="\n\n")
```

Obliczenie i wyświetlenie macierzy sztywności elementów

```
K1 = Ke(E * A1, L1)
K2 = Ke(E * A2, L2)

print(f"{K1} = K1", end="\n\n")
print(f"{K2} = K2", end="\n\n")
```

Obliczenie i wyświetlenie równoważników węzłowych obciążenia

```
P1 = Pe(q, L1)
P2 = Pe(q, L2)

print(f"{P1} = P1", end="\n\n")
print(f"{P2} = P2", end="\n\n")
```

Agregacja i wyświetlenie macierzy sztywności

```
K = B1.T.dot(K1).dot(B1) + B2.T.dot(K2).dot(B2)

print(f"{K} = K", end="\n\n")
```

Agregacja i wyświetlenie równoważników węzłowych obciążenia

```
P = B1.T.dot(P1) + B2.T.dot(P2)

print(f"{P} = P", end="\n\n")
```

Budowa i wyświetlenie wektora sił węzłowych

```
F = np.zeros((ln * lq, 1))
F[1] = T1
F[2] = T2

print(f"{F} = F", end="\n\n")
```

Uwzględnienie warunków brzegowych, wyświetlanie macierzy

- Domyślnie przy przypisywaniu tablic, Python nie tworzy ich kopii tylko referencje, dlatego jawnie kopiujemy tablice do nowych zmiennych.

```
Kw = np.copy(K)
Fw = np.copy(F)
Pw = np.copy(P)

Kw[0, :] = 0
Kw[:, 0] = 0
Kw[0, 0] = 1

Fw[0] = 0
Pw[0] = 0

print(f"{Kw} = Kw", end="\n\n")
print(f"{Fw} = Fw", end="\n\n")
print(f"{Pw} = Pw", end="\n\n")
```

Rozwiązanie układu równań i wyświetlenie przemieszczeń

```
Q = np.linalg.solve(Kw, Pw + Fw)

print(f"{Q} = Q", end="\n\n")
```

Obliczenie i wyświetlenie reakcji

```
R = np.dot(K, Q) - P

print(f"{R} = R", end="\n\n")
```

Obliczenie i wyświetlenie wektorów przemieszczeń dla elementów

```
Q1 = np.dot(B1, Q)
Q2 = np.dot(B2, Q)

print(f"{Q1} = Q1", end="\n\n")
print(f"{Q2} = Q2", end="\n\n")
```

Obliczenie i wyświetlenie wektorów sił węzłowych w elementach z warunków równowagi elementów

```
F1 = np.dot(K1, Q1) - P1
F2 = np.dot(K2, Q2) - P2

print(f"{F1} = F1", end="\n\n")
print(f"{F2} = F2", end="\n\n")
```

Definicja wektora funkcji kształtu

```
def dN(x, L):
  return np.array([[ -1 / L, 1 / L ]])
```

Definicja wektora pochodnych funkcji kształtu

```
def d1N(x, L):
  return np.array([[ -1 / L, 1 / L ]])
```

Definicja funkcji przemieszczeń dla elementu

- Wszystkie parametry funkcji są skalarami, ze względu na późniejszą wektoryzację funkcji.

```
def V(x, L, q1, q2, E, A):
  return E * A * np.dot(d1N(x, L), [[q1], [q2]])
```

Definicja funkcji naprężeń dla elementu

```
def s(x, L, q1, q2, E):
  return E * np.dot(d1N(x, L), [[q1], [q2]])
```

Obliczenie i wyświetlenie wektorów sił węzłowych w elementach z wykorzystaniem funkcji kształtu

```
F1N = np.vstack((V(0, L1, Q1[0, 0], Q1[1, 0], E, A1), V(L1, L1, Q1[0, 0], Q1[1, 0], E, A1)))
F2N = np.vstack((V(0, L2, Q2[0, 0], Q2[1, 0], E, A2), V(L2, L2, Q2[0, 0], Q2[1, 0], E, A2)))

print(f"{F1N} = F1N", end="\n\n")
print(f"{F2N} = F2N", end="\n\n")
```

Porównaj wartości sił przekrojowych w węzłach elementów, obliczonych z wykorzystaniem:

- równań równowagi elementów,

- zróżniczkowanych funkcji kształtu.

Jak poprawić wyniki?

Wektoryzacja funkcji przemieszczeń, siły normalnej i naprężeń

- Wektoryzacja pozwoli na prowadzenie obliczeń dla zestawu danych w tablicy i nie będzie wymagała iterowania po elementach tablicy. Jest to bardzo wydajna technika w stosunku do pętli.

```
vecu = np.vectorize(u)
vecV = np.vectorize(V)
vecs = np.vectorize(s)
```

Generowanie współrzędnych dla elementów

```
x1 = np.linspace(0, L1, 5)
x2 = np.linspace(0, L2, 5)
```

Obliczenie wartości przemieszczeń w elementach

```
u1 = vecu(x1, L1, Q1[0, 0], Q1[1, 0])
u2 = vecu(x2, L2, Q2[0, 0], Q2[1, 0])
```

Obliczenie wartości siły normalnej w elementach

```
V1 = vecV(x1, L1, Q1[0, 0], Q1[1, 0], E, A1)
V2 = vecV(x2, L2, Q2[0, 0], Q2[1, 0], E, A2)
```

Obliczenie wartości naprężeń normalnych w elementach

```
s1 = vecs(x1, L1, Q1[0, 0], Q1[1, 0], E)
s2 = vecs(x2, L2, Q2[0, 0], Q2[1, 0], E)
```

Wykresy

```
mpl.pyplot.clf()

mpl.pyplot.subplot(3, 1, 1)
mpl.pyplot.plot(x1, u1, label = "element 1", linewidth = 3)
mpl.pyplot.plot(x2 + L1, u2, label = "element 2", linewidth = 3)
mpl.pyplot.title("Wykres przemieszczenia")
mpl.pyplot.legend()
mpl.pyplot.xlabel("x1, x2 + L1")
mpl.pyplot.ylabel("u1(x1), u2(x2)")
mpl.pyplot.xlim(0, 1.2)
mpl.pyplot.ylim(0, 0.35e-3)
mpl.pyplot.grid()

mpl.pyplot.subplot(3, 1, 2)
mpl.pyplot.plot(x1, V1, label = "element 1", linewidth = 3)
mpl.pyplot.plot(x2 + L1, V2, label = "element 2", linewidth = 3)
mpl.pyplot.title("Wykres sily normalnej")
mpl.pyplot.legend()
mpl.pyplot.xlabel("x1, x2 + L1")
mpl.pyplot.ylabel("V1(x1), V2(x2)")
mpl.pyplot.xlim(0, 1.2)
mpl.pyplot.ylim(0, 60e3)
mpl.pyplot.grid()

mpl.pyplot.subplot(3, 1, 3)
mpl.pyplot.plot(x1, s1, label = "element 1", linewidth = 3)
```

```
mpl.pyplot.plot(x2 + L1, s2, label = "element 2", linewidth = 3)
mpl.pyplot.title("Wykres naprezen normalnych")
mpl.pyplot.legend()
mpl.pyplot.xlabel("x1, x2 + L1")
mpl.pyplot.ylabel("s1(x1), s2(x2)")
mpl.pyplot.xlim(0, 1.2)
mpl.pyplot.ylim(0, 120e6)
mpl.pyplot.grid()

mpl.pyplot.tight_layout()
```